Even Faster Elastic-Degenerate String Matching via Fast Matrix Multiplication

Giulia Bernardini, Pawel Gawrychowski, Nadia Pisanti, Solon Pissis, Giovanna Rosone

Even Faster Elastic-Degenerate String Matching via Fast Matrix Multiplication

Giulia Bernardini
Department of Informatics, Systems and Communication, University of Milano - Bicocca, Italy
giulia.bernardini@unimib.it

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland
gawry@cs.uni.wroc.pl

Nadia Pisanti
Department of Computer Science, University of Pisa, Italy
ERABLE Team, INRIA, France
pisanti@di.unipi.it

Solon P. Pissis
CWI, Amsterdam, The Netherlands
solon.pissis@cwi.nl

Giovanna Rosone
Department of Computer Science, University of Pisa, Italy
giovanna.rosone@unipi.it

Abstract

An elastic-degenerate (ED) string is a sequence of n sets of strings of total length N, which was recently proposed to model a set of similar sequences. The ED string matching (EDSM) problem is to find all occurrences of a pattern of length m in an ED text. The EDSM problem has recently received some attention in the combinatorial pattern matching community, and an $\mathcal{O}(nm^{1.5}\sqrt{\log m}+N)$-time algorithm is known [Aoyama et al., CPM 2018]. The standard assumption in the prior work on this question is that N is substantially larger than both n and m, and thus we would like to have a linear dependency on the former. Under this assumption, the natural open problem is whether we can decrease the 1.5 exponent in the time complexity, similarly as in the related (but, to the best of our knowledge, not equivalent) *word break* problem [Backurs and Indyk, FOCS 2016].

Our starting point is a conditional lower bound for the EDSM problem. We use the popular combinatorial Boolean matrix multiplication (BMM) conjecture stating that there is no truly subcubic *combinatorial* algorithm for BMM [Abboud and Williams, FOCS 2014]. By designing an appropriate reduction we show that a combinatorial algorithm solving the EDSM problem in $\mathcal{O}(nm^{1.5-\epsilon}+N)$ time, for any $\epsilon > 0$, refutes this conjecture. Of course, the notion of combinatorial algorithms is not clearly defined, so our reduction should be understood as an indication that decreasing the exponent requires fast matrix multiplication.

Two standard tools used in algorithms on strings are string periodicity and fast Fourier transform. Our main technical contribution is that we successfully combine these tools with fast matrix multiplication to design a non-combinatorial $\mathcal{O}(nm^{1.381}+N)$-time algorithm for EDSM. To the best of our knowledge, we are the first to do so.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases string algorithms, pattern matching, elastic-degenerate string, matrix multiplication, fast Fourier transform

Digital Object Identifier 10.4230/LIPIcs.ICALP.2019.21

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `https://arxiv.org/abs/1905.02298`.

46th International Colloquium on Automata, Languages, and Programming (ICALP 2019).
Editors: Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi;
Article No. 21; pp. 21:1–21:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding GR and NP are partially supported by MIUR-SIR project CMACBioSeq "Combinatorial methods for analysis and compression of biological sequences" grant n. RBSI146R5L.

1 Introduction

Boolean matrix multiplication (BMM) is one of the most fundamental computational problems. Apart from its theoretical interest, it has a wide range of applications [37, 52, 29, 27, 46]. BMM is also the core combinatorial part of integer matrix multiplication. In both problems, we are given two $\mathcal{N} \times \mathcal{N}$ matrices and we are to compute $\mathcal{N}^2$ values. Integer matrix multiplication can be performed in *truly subcubic* time, *i.e.*, in $\mathcal{O}(\mathcal{N}^{3-\epsilon})$ operations over the field, for some $\epsilon > 0$. The fastest known algorithms for this problem run in $\mathcal{O}(\mathcal{N}^{2.373})$ time [30, 54]. These algorithms are known as *algebraic*: they rely on the underlying ring structure.

There also exists a different family of algorithms for the BMM problem known as *combinatorial*. Their focus is on unveiling the combinatorial structure in the Boolean matrices to reduce redundant computations. A series of results [7, 9, 15] culminating in an $\hat{\mathcal{O}}(\mathcal{N}^3/\log^4 \mathcal{N})$-time algorithm [58] (the $\hat{\mathcal{O}}(\cdot)$ notation suppresses poly(log log) factors) has led to the popular *combinatorial BMM conjecture* stating that there is no combinatorial algorithm for BMM working in time $\mathcal{O}(\mathcal{N}^{3-\epsilon})$, for any $\epsilon > 0$ [2]. There has been ample work on applying this conjecture to obtain *BMM hardness* results: see, *e.g.*, [44, 2, 49, 33, 43, 42, 17].

String matching is another fundamental problem. The problem is to find all fragments of a string *text* of length n that match a string *pattern* of length m. This problem has several linear-time solutions [22]. In many real-world applications, it is often the case that letters at some positions are either unknown or uncertain. A way of representing these positions is with a subset of the alphabet Σ. Such a representation is called *degenerate string*. The first efficient algorithm for a degenerate text and a standard pattern was published by Fischer and Paterson in 1974 [28]. It has undergone several improvements since then [36, 39, 20, 19]. The first efficient algorithm for a degenerate pattern and a standard text was published by Abrahamson in 1987 [3], followed by several practically efficient algorithms [57, 47, 34].

Degenerate letters are used in the IUPAC notation [38] to represent a position in a DNA sequence that can have multiple possible alternatives. These are used to encode the consensus of a population of sequences [21, 4] in a multiple sequence alignment (MSA). In the presence of insertions or deletions in the MSA, we may need to consider alternative representations. Consider the following MSA of three closely-related sequences (on the left):

```
GCAACGGGTA--TT
GCAACGGGTATATT
GCACCTGG----TT
```

$$\tilde{T} = \{\ \texttt{GCA}\ \} \cdot \left\{ \begin{array}{c} \texttt{A} \\ \texttt{C} \end{array} \right\} \cdot \{\ \texttt{C}\ \} \cdot \left\{ \begin{array}{c} \texttt{G} \\ \texttt{T} \end{array} \right\} \cdot \{\ \texttt{GG}\ \} \cdot \left\{ \begin{array}{c} \texttt{TA} \\ \texttt{TATA} \\ \varepsilon \end{array} \right\} \cdot \{\ \texttt{TT}\ \}$$

These sequences can be compacted into a single sequence $\tilde{T}$ of sets of strings (on the right) containing some deterministic and some *non-deterministic* segments. A non-deterministic segment is a finite set of deterministic strings and may contain the empty string ε corresponding to a deletion. The total number of segments is the *length* of $\tilde{T}$ and the total number of letters is the *size* of $\tilde{T}$. We denote the length by $n = |\tilde{T}|$ and the size by $N = ||\tilde{T}||$.

This representation has been defined in [35] by Iliopoulos et al. as an *elastic-degenerate* (ED) string. Being a sequence of subsets of Σ^*, it can be seen as a generalization of a degenerate string. The natural problem that arises is finding all matches of a deterministic pattern P in an ED text $\tilde{T}$. This is the *elastic-degenerate string matching* (EDSM) problem. Since its introduction in 2017 [35], it has attracted some attention in the combinatorial pattern matching community, and a series of results have been published. The simple

algorithm by Iliopoulos et al. [35] for EDSM was first improved by Grossi et al. in the same year, who showed that, for a pattern of length m, the EDSM problem can be solved *on-line* in $\mathcal{O}(nm^2 + N)$ time [32]; on-line means that the text is read segment-by-segment and an occurrence is detected as soon as possible. This result was improved by Aoyama et al. [6] who presented an $\mathcal{O}(nm^{1.5}\sqrt{\log m} + N)$-time algorithm. An important feature of these bounds is their *linear dependency* on N. A different branch of on-line algorithms waiving the linear-dependency restriction exists [32, 48, 18]. Moreover, the EDSM problem has been considered under Hamming and edit distance [12].

A question with a somewhat similar flavor is the *word break* problem. We are given a dictionary $\mathcal{D}$, $m = ||\mathcal{D}||$, and a string S, $n = |S|$, and the question is whether we can split S into fragments that appear in $\mathcal{D}$ (the same element of $\mathcal{D}$ can be used multiple times). Backurs and Indyk [8] designed an $\tilde{\mathcal{O}}(nm^{1/2-1/18}+m)$-time algorithm for this problem (the $\tilde{\mathcal{O}}$ notation suppresses poly(log) factors). Bringmann et al. [14] improved this to $\tilde{\mathcal{O}}(nm^{1/3} + m)$ and showed that this is optimal for combinatorial algorithms by a reduction from k-Clique. Their algorithm uses fast Fourier transform (FFT), and so it is not clear whether it should be considered combinatorial. While this problem seems similar to EDSM, there does not seem to be a direct reduction and so their lower bound does not immediately apply.

Our Results. It is known that BMM and triangle detection in graphs either both have truly subcubic combinatorial algorithms or none of them do [56]. Recall also that the currently fastest algorithm with linear dependency on N for the EDSM problem runs in $\mathcal{O}(nm^{1.5}\sqrt{\log m} + N)$ time [6]. In this paper we prove the following two theorems.

▶ **Theorem 1.** *If the EDSM problem can be solved in $\mathcal{O}(nm^{1.5-\epsilon} + N)$ time, for any $\epsilon > 0$, with a combinatorial algorithm, then there exists a truly subcubic combinatorial algorithm for triangle detection.*

Arguably, the notion of combinatorial algorithms is not clearly defined, and Theorem 1 should be understood as an indication that in order to achieve a better complexity one should use fast matrix multiplication. Indeed, there are examples where a lower bound conditioned on BMM was helpful in constructing efficient algorithms using fast matrix multiplication [1, 16, 13, 45, 24, 55, 59]. We successfully design such a non-combinatorial algorithm by combining three ingredients: a string periodicity argument, FFT, and fast matrix multiplication. While periodicity is the usual tool in combinatorial pattern matching [40, 23, 41] and using FFT is also not unusual (for example, it often shows up in approximate string matching [3, 5, 19, 31]), to the best of our knowledge, we are the first to combine these with fast matrix multiplication. Specifically, we show the following result for the EDSM problem.

▶ **Theorem 2.** *The EDSM problem can be solved on-line in expected $\mathcal{O}(nm^{1.381} + N)$ time.*

An important building block in our solution that might find applications in other problems is a method of selecting a small set of length-ℓ substrings of the pattern, called *anchors*, so that any relevant occurrence of a string from an ED text set contains at least one but not too many such anchors inside. This is obtained by rephrasing the question in a graph-theoretical language and then generalizing the well-known fact that an instance of the *hitting set* problem with m sets over $[n]$, each of size at least k, has a solution of size $\mathcal{O}(n/k \cdot \log m)$. While the idea of carefully selecting some substrings of the same length is not new, for example Kociumaka et al. [41] used it to design a data structure for pattern matching queries on a string, our setting is different and hence so is the method of selecting these substrings.

Roadmap. Section 2 provides the necessary definitions and notation as well the algorithmic toolbox used throughout the paper. In Section 3 we prove our hardness result for the EDSM problem (Theorem 1). In Section 4 we present our algorithm for the same problem (Theorem 2); this is the most technically involved part of the paper.

2 Preliminaries

Let $T = T[1]T[2]\dots T[n]$ be a *string* of length $|T| = n$ over a finite ordered alphabet Σ of size $|\Sigma| = \sigma$. For two positions i and j on T, we denote by $T[i\,..\,j] = T[i]\dots T[j]$ the *substring* of T that starts at position i and ends at position j (it is of length 0 if $j < i$). By ε we denote the *empty string* of length 0. A *prefix* of T is a substring of the form $T[1\,..\,j]$, and a suffix of T is a substring of the form $T[i\,..\,n]$. T^r denotes the *reverse* of T, that is, $T[n]T[n-1]\dots T[1]$. We say that a string X is a *power* of a string Y if there exists an integer $k > 1$, such that X is expressed as k consecutive concatenations of Y, denoted by $X = Y^k$. A period of a string X is any integer $p \in [1, |X|]$ such that $X[i] = X[i+p]$ for every $i = 1, 2, \dots, |X| - p$, and *the period*, denoted by $\mathrm{per}(X)$, is the smallest such p. We call a string X *strongly periodic* if $\mathrm{per}(X) \le |X|/4$.

▶ **Lemma 3** ([26]). *If p and q are both periods of the same string X, and additionally $p + q \le |X| + 1$, then $\gcd(p, q)$ is also a period of X.*

A *trie* is a rooted tree in which every edge is labeled with a single letter, and every two edges outgoing from the same node have different labels. The label of a node u in such a tree T, denoted by $\mathcal{L}(u)$, is defined as the concatenation of the labels of all the edges on the path from the root of T to u. Thus, the label of the root of T is ε, and a trie is a representation of a set of strings consisting of the labels of all its leaves. By replacing each path p consisting of nodes with exactly one child by an edge labeled by the concatenation of the labels of the edges of p we obtain a *compact trie*. The nodes of the trie that are removed after this transformation are called *implicit*, while the remaining ones are referred to as *explicit*. The *suffix tree* of a string S is the compact trie representing all suffixes of $S\$$, $\$ \notin \Sigma$, where instead of explicitly storing the label $S[i\,..\,j]$ of an edge we represent it by a pair (i, j).

A *heavy path decomposition* of a tree T is obtained by selecting, for every non-leaf node $u \in T$, its child v such that the subtree rooted at v is the largest. This decomposes the nodes of T into node-disjoint paths, with each such path p (called a heavy path) starting at some node, called the *head* of p, and ending at a leaf. An important property of such a decomposition is that the number of distinct heavy paths above any leaf (that is, intersecting the path from a leaf to the root) is only logarithmic in the size of T [51].

Let $\tilde{\Sigma}$ denote the set of all finite non-empty subsets of Σ^*. Previous works (cf. [35, 32, 12, 6, 48]) define $\tilde{\Sigma}$ as the set of all finite non-empty subsets of Σ^* excluding $\{\varepsilon\}$ but we waive here the latter restriction as it has no algorithmic implications. An *elastic-degenerate string*, or ED string, over alphabet Σ, is a string over $\tilde{\Sigma}$, i.e., an ED string is an element of $\tilde{\Sigma}^*$.

Let $\tilde{T}$ denote an ED string of *length* n, i.e. $|\tilde{T}| = n$. We assume that for any $1 \le i \le n$, the set $\tilde{T}[i]$ is implemented as an array and can be accessed by an index, i.e., $\tilde{T}[i] = \{\tilde{T}[i][k] \mid k = 1, \dots, |\tilde{T}[i]|\}$. For any $\tilde{c} \in \tilde{\Sigma}$, $||\tilde{c}||$ denotes the total length of all strings in $\tilde{c}$, and for any ED string $\tilde{T}$, $||\tilde{T}||$ denotes the total length of all strings in all $\tilde{T}[i]$s or the *size* of $\tilde{T}$, i.e., $||\tilde{c}|| = \sum_{s\in\tilde{c}} |s|$ and $||\tilde{T}|| = \sum_{i=1}^{n} ||\tilde{T}[i]||$. An ED string $\tilde{T}$ can be thought of as a representation of the set of strings $\mathcal{A}(\tilde{T}) = \tilde{T}[1] \times \dots \times \tilde{T}[n]$, where $A \times B = \{xy \mid x \in A, y \in B\}$ for any sets of strings A and B. For any ED string $\tilde{X}$ and a pattern P, we say that P *matches* $\tilde{X}$ if

1. $|\tilde{X}| = 1$ and P is a substring of some string in $\tilde{X}[1]$, or,
2. $|\tilde{X}| > 1$ and $P = P_1 \dots P_{|\tilde{X}|}$, where P_1 is a suffix of some string in $\tilde{X}[1]$, $P_{|\tilde{X}|}$ is a prefix of some string in $\tilde{X}[|\tilde{X}|]$, and $P_i \in \tilde{X}[i]$, for all $1 < i < |\tilde{X}|$.

We say that an *occurrence* of a string P *ends* at position j of an ED string $\tilde{T}$ if there exists $i \leq j$ such that P matches $\tilde{T}[i] \dots \tilde{T}[j]$. We will refer to string P as the *pattern* and to ED string $\tilde{T}$ as the *text*. We define the main problem considered in this paper.

Elastic-Degenerate String Matching (EDSM)
INPUT: A string P of length m and an ED string $\tilde{T}$ of length n and size $N \geq m$.
OUTPUT: All positions in $\tilde{T}$ where at least one occurrence of P ends.

▶ **Example 4.** Pattern $P = \texttt{GTAT}$ ends at positions 2, 6, and 7 of the following text $\tilde{T}$.

$$\tilde{T} = \{\ \texttt{ATGTA}\ \} \cdot \left\{ \begin{matrix} \texttt{A} \\ \texttt{T} \end{matrix} \right\} \cdot \{\ \texttt{C}\ \} \cdot \left\{ \begin{matrix} \texttt{G} \\ \texttt{T} \end{matrix} \right\} \cdot \{\ \texttt{CG}\ \} \cdot \left\{ \begin{matrix} \texttt{TA} \\ \texttt{TATA} \\ \varepsilon \end{matrix} \right\} \cdot \left\{ \begin{matrix} \texttt{TATGC} \\ \texttt{TTTTA} \end{matrix} \right\}$$

Aoyama et al. [6] obtained an on-line $\mathcal{O}(nm^{1.5}\sqrt{\log m} + N)$-time algorithm by designing an efficient solution for the following problem.

Active Prefixes (AP)
INPUT: A string P of length m, a bit vector U of size m, a set $\mathcal{S}$ of strings of total length N.
OUTPUT: A bit vector V of size m with $V[j] = 1$ if and only if there exists $S \in \mathcal{S}$ and $i \in [1, m], U[i] = 1$, such that $P[1\,..\,i] \cdot S = P[1\,..\,i + |S|]$ and $j = i + |S|$.

In more detail, given an ED text one should consider an instance of the AP problem per segment. Hence, an $\mathcal{O}(f(m) + N_i)$ solution for AP (with N_i being the size of the i-th segment of the ED text) implies an $\mathcal{O}(n \cdot f(m) + N)$ solution for EDSM, as $N = \sum_{i=1}^{n} N_i$. We provide an example of the AP problem.

▶ **Example 5.** Let $P = \texttt{ababbababab}$ of length $m = 11$, $U = 01000100000$, and $\mathcal{S} = \{\varepsilon, \texttt{ab}, \texttt{abb}, \texttt{ba}, \texttt{baba}\}$. We have that $V = 01011101010$.

For our hardness results we rely on BMM and the following closely related problem.

Boolean Matrix Multiplication (BMM)
INPUT: Two $\mathcal{N} \times \mathcal{N}$ Boolean matrices A and B.
OUTPUT: $\mathcal{N} \times \mathcal{N}$ Boolean matrix C, where $C[i, j] = \bigvee_k (A[i, k] \wedge B[k, j])$.

Triangle Detection (TD)
INPUT: Three $\mathcal{N} \times \mathcal{N}$ Boolean matrices A, B and C.
OUTPUT: Are there i, j, k such that $A[i, j] = B[j, k] = C[k, i] = 1$?

An algorithm is called *truly subcubic* if it runs in $\mathcal{O}(\mathcal{N}^{3-\epsilon})$ time, for some $\epsilon > 0$. TD and BMM either both have truly subcubic combinatorial algorithms, or none of them do [56].

3 EDSM Conditional Lower Bound

We show a conditional lower bound for the EDSM problem. Specifically, we show that TD can be reduced to the EDSM problem. We work with the decision version: the goal is to detect whether there exists at least one occurrence of P in $\tilde{T}$.

▶ **Theorem 1.** *If the EDSM problem can be solved in $\mathcal{O}(nm^{1.5-\epsilon}+N)$ time, for any $\epsilon>0$, with a combinatorial algorithm, then there exists a truly subcubic combinatorial algorithm for triangle detection.*

Proof. Consider an instance of TD, where we are given three $\mathcal{N}\times\mathcal{N}$ Boolean matrices A, B, C, and the question is to check if there exist i, j, k such that $A[i, j] = B[j, k] = C[k, i] = 1$. Let s be a parameter to be determined later that corresponds to decomposing B into blocks of size $(\mathcal{N}/s)\times(\mathcal{N}/s)$. We reduce to an instance of EDSM over an alphabet Σ of size $\mathcal{O}(\mathcal{N})$.

Pattern P. We construct P by concatenating, in some fixed order, the following strings:

$$P(i,x,y) = v(i)xa^{\mathcal{N}/s}x\$\$ya^{\mathcal{N}/s}yv(i)$$

for every $i=1,2,\ldots,\mathcal{N}$ and $x,y=1,2,\ldots,s$, where $a\in\Sigma_1$, $\$\in\Sigma_2$, $x\in\Sigma_3$, $y\in\Sigma_4$, $v(i)\in\Sigma_5$, and $\Sigma_1,\Sigma_2,\ldots,\Sigma_5$ are disjoint subsets of Σ.

ED text $\tilde{T}$. The text $\tilde{T}$ consists of three parts. Its middle part encodes all the entries equal to 1 in matrices A, B and C, and consists of three string sets $\mathcal{X}=\mathcal{X}_1\cdot\mathcal{X}_2\cdot\mathcal{X}_3$, where:

1. $\mathcal{X}_1$ contains all strings of the form $v(i)xa^j$, for some $i=1,2,\ldots,\mathcal{N}$, $x=1,2,\ldots,s$ and $j=1,2,\ldots,\mathcal{N}/s$ such that $A[i,(x-1)\cdot(\mathcal{N}/s)+j]=1$;
2. $\mathcal{X}_2$ contains all strings of the form $a^{\mathcal{N}/s-j}\ x\$\$ya^{\mathcal{N}/s-k}$, for some $x,y=1,2,\ldots,s$ and $j,k=1,2,\ldots,\mathcal{N}/s$ such that $B[(x-1)\cdot(\mathcal{N}/s)+j,(y-1)\cdot(\mathcal{N}/s)+k]=1$, i.e., if the corresponding entry of B is 1;
3. $\mathcal{X}_3$ contains all strings of the form $a^kyv(i)$, for some $i=1,2,\ldots,\mathcal{N}$, $y=1,2,\ldots,s$ and $k=1,2,\ldots,\mathcal{N}/s$ such that $C[(y-1)\cdot(\mathcal{N}/s)+k,i]=1$.

It is easy to see that $|P(i,x,y)|=\mathcal{O}(\mathcal{N}/s)$. This implies the following:

1. The length of the pattern is $m=\mathcal{O}(\mathcal{N}\cdot s^2\cdot\mathcal{N}/s)=\mathcal{O}(\mathcal{N}^2\cdot s)$;
2. The size of $\mathcal{X}$ is $||\mathcal{X}||=\mathcal{O}(\mathcal{N}\cdot s\cdot\mathcal{N}/s\cdot\mathcal{N}/s+s^2\cdot(\mathcal{N}/s)^2\cdot\mathcal{N}/s+\mathcal{N}\cdot s\cdot\mathcal{N}/s\cdot\mathcal{N}/s)=\mathcal{O}(\mathcal{N}^3/s)$.

By the above construction, we obtain the following fact.

▶ **Fact 6.** $P(i,x,y)$ matches $\mathcal{X}$ if and only if the following holds for some $j,k=1,2,\ldots,\mathcal{N}/s$:

$$A[i,(x-1)\cdot(\mathcal{N}/s)+j]=B[(x-1)\cdot(\mathcal{N}/s)+j,(y-1)\cdot(\mathcal{N}/s)+k]=C[(y-1)\cdot(\mathcal{N}/s)+k,i]=1.$$

Solving the TD problem thus reduces to taking the disjunction of all such conditions. Let us write down all strings $P(i,x,y)$ in some arbitrary but fixed order to obtain $P=P_1P_2\ldots P_z$ with $z=\mathcal{N}s^2$, where every $P_t=P(i,x,y)$, for some i,x,y. We aim to construct a small number of sets of strings that, when considered as an ED text, match any prefix $P_1P_2\ldots P_t$ of the pattern, $1\le t\le z-1$; a similar construction can be carried on to obtain sets of strings that match any suffix $P_k\ldots P_{z-1}P_z$, $2\le k\le z$. These sets will then be added to the left and to the right of $\mathcal{X}$, respectively, to obtain the ED text $\tilde{T}$.

ED Prefix. We construct $\log z$ sets of strings as follows. The first one contains the empty string ε and $P_1P_2\ldots P_{z/2}$. The second one contains ε, $P_1P_2\ldots P_{z/4}$ and $P_{z/2+1}\ldots P_{z/2+z/4}$. The third one contains ε, $P_1P_2\ldots P_{z/8}$, $P_{z/4+1}\ldots P_{z/4+z/8}$, $P_{z/2+1}\ldots P_{z/2+z/8}$ and $P_{z/2+z/4+1}\ldots P_{z/2+z/4+z/8}$. Formally, for every $i=1,2,\ldots,\log z$, the i-th of such sets is:

$$\tilde{T}_i^p=\varepsilon\cup\{P_{j\frac{z}{2^{i-1}}+1}\ldots P_{j\frac{z}{2^{i-1}}+\frac{z}{2^i}}\mid j=0,1,\ldots,2^{i-1}-1\}.$$

ED Suffix. We similarly construct $\log z$ sets to be appended to $\mathcal{X}$:

$$\tilde{T}_i^s = \varepsilon \cup \{P_{z-j\frac{z}{2^{i-1}}-\frac{z}{2^i}+1} \dots P_{z-j\frac{z}{2^{i-1}}} \mid j = 0, 1, \dots, 2^{i-1}-1\}.$$

The total length of all the ED prefix and ED suffix strings is $\mathcal{O}(\log z \cdot \mathcal{N}^2 \cdot s) = \mathcal{O}(\mathcal{N}^2 \cdot s \cdot \log \mathcal{N})$. The whole ED text $\tilde{T}$ is: $\tilde{T} = \tilde{T}_1^p \cdot \dots \cdot \tilde{T}_{\log z}^p \cdot \mathcal{X} \cdot \tilde{T}_{\log z}^s \cdot \dots \cdot \tilde{T}_1^s$.

▶ **Lemma 7.** *The pattern P occurs in the ED text $\tilde{T}$ if and only if there exist i, j, k such that $A[i, j] = B[j, k] = C[k, i] = 1$.*

Proof. By Fact 6, if such i, j, k exist then P_t matches $\mathcal{X}$, for some $t \in \{1, \dots, z\}$. Then, by construction of the sets $\tilde{T}_i^p$ and $\tilde{T}_i^s$, the prefix $P_1 \dots P_{t-1}$ matches the ED prefix (this can be proved by induction), and similarly the suffix $P_{t+1} \dots P_z$ matches the ED suffix, so the whole P matches $\tilde{T}$, and so P occurs in $\tilde{T}$. Because of the letters \$ appearing only in the center of P_is and strings from $\mathcal{X}_2$, every P_i and a concatenation of $X_1 \in \mathcal{X}_1$, $X_2 \in \mathcal{X}_2$, $X_3 \in \mathcal{X}_3$ having the same length, and the P_is being distinct, there is an occurrence of the pattern P in $\tilde{T}$ if and only if $X_1X_2X_3 = P_t$ for some t and $X_1 \in \mathcal{X}_1$, $X_2 \in \mathcal{X}_2$, $X_3 \in \mathcal{X}_3$. But then, by Fact 6 there exists a triangle. ◀

Note that for the EDSM problem we have $m = \mathcal{N}^2 \cdot s$, $n = 1 + 2\log z$ and $N = ||\mathcal{X}|| + \mathcal{O}(\mathcal{N}^2 \cdot \dots \cdot \log \mathcal{N})$. Thus if we had a solution running in $\mathcal{O}(\log z \cdot m^{1.5-\epsilon} + ||\mathcal{X}|| + \mathcal{N}^2 \cdot s \cdot \log \mathcal{N}) = \mathcal{O}(\log \mathcal{N} \cdot (\mathcal{N}^2 \cdot s)^{1.5-\epsilon} + \mathcal{N}^3/s)$ time, for some $\epsilon > 0$, by choosing a sufficiently small $\alpha > 0$ and setting $s = \mathcal{N}^\alpha$ we would obtain an $\mathcal{O}(\mathcal{N}^{3-\delta})$-time algorithm for TD, for some $\delta > 0$. ◀

4 An $\mathcal{O}(nm^{1.381} + N)$-time Algorithm for EDSM

Our goal is to design a non-combinatorial $\mathcal{O}(nm^{1.381} + N)$-time algorithm for EDSM. It suffices to solve an instance of the AP problem in $\mathcal{O}(m^{1.381} + N)$ time. We further reduce the AP problem to a logarithmic number of restricted instances of the problem, in which the length of every string $S \in \mathcal{S}$ is in $[(10/9)^k, (10/9)^{k+1})$, for $k = 0, \dots, \log m / \log(10/9)$. If we solve every such instance in $\mathcal{O}(f(m) + N)$ time, then we can solve the original instance in $\mathcal{O}(f(m) \log m + N)$ time by taking the disjunction of results. We partition the strings in $\mathcal{S}$ into three types, compute the corresponding bit vector V for each type separately and in different ways, and, finally, take the disjunction to obtain the answer for the restricted instance.

Partitioning S. Let $\ell = 8/9 \cdot (10/9)^k$ (to avoid clutter we assume that ℓ is an integer, but this can be avoided by appropriately adjusting the constants), so that the length of every string in S belongs to $[9/8 \cdot \ell, 5/4 \cdot \ell)$. The three types of strings are as follows:

Type 1: Strings $S \in \mathcal{S}$ such that every length-ℓ substring of S is not strongly periodic.

Type 2: Strings $S \in \mathcal{S}$ containing at least one length-ℓ substring that is not strongly periodic and at least one length-ℓ substring that is strongly periodic.

Type 3: Strings $S \in \mathcal{S}$ such that every length-ℓ substring of S is strongly periodic (in Lemma 8 we show that in this case $\mathrm{per}(S) \le \ell/4$).

These three types are evidently a partition of $\mathcal{S}$ and, before we proceed with the algorithm, we need to show that we can determine the type of a string $S \in \mathcal{S}$ in $\mathcal{O}(|S|)$ time. We start with showing that, in fact, strings of type 3 are exactly strings with period at most $\ell/4$.

▶ **Lemma 8.** *Let S be a string. If $\mathrm{per}(S[j \,..\, j + \ell - 1]) \le \ell/4$ for every j then $\mathrm{per}(S) \le \ell/4$.*

▶ **Lemma 9.** *Given a string S we can determine its type in $\mathcal{O}(|S|)$ time.*

4.1 Type 1 Strings

In this section we show how to solve a restricted instance of the AP problem where every string $S \in \mathcal{S}$ is of type 1, that is, each of its length-ℓ substrings is not strongly periodic, and furthermore $|S| \in [9/8 \cdot \ell, 5/4 \cdot \ell)$ for some $\ell \leq m$. Observe that all (hence at most $1/4 \cdot \ell$) length-ℓ substrings of any $S \in \mathcal{S}$ must be distinct, as otherwise we would be able to find two occurrences of a length-ℓ substring at distance at most $1/4 \cdot \ell$ in S, making the period of the substring at most $1/4 \cdot \ell$ and contradicting the assumption that S is of type 1.

We start with constructing the suffix tree ST of P (our pattern in the EDSM problem) in $\mathcal{O}(m \log m)$ time [53] (note that we are spending $\mathcal{O}(m \log m)$ time and not just $\mathcal{O}(m)$ as to avoid any assumptions on the alphabet). For every explicit node $u \in ST$, we construct a perfect hash function mapping the first letter on every edge outgoing from u to the corresponding edge. This takes $\mathcal{O}(m \log m)$ time [50] and allows us to navigate in ST in constant time per letter. Then, for every $S \in \mathcal{S}$ we check if it occurs in P using the suffix tree in $\mathcal{O}(|S|)$ time, and if not disregard it from further consideration. We want to further partition $\mathcal{S}$ into $\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_{\log m}$ that are processed separately. For every $\mathcal{S}_k$, we want to select a set of length-ℓ substrings of P, called the *anchors*, each represented by one of its occurrences in P, such that:

1. The total number of occurrences of all anchors in P is $\mathcal{O}(m/\ell \cdot \log^2 m)$.
2. For every $S \in \mathcal{S}_k$, at least one of its length-ℓ substrings is an anchor.
3. For every $S \in \mathcal{S}_k$, at most $\mathcal{O}(\log^2 m)$ of its length-ℓ substrings are anchors.

We formalize this using the following auxiliary problem, which is a strengthening of the *hitting set* problem: for any collection of m sets over $[n]$, each of size at least k, we can choose a subset of $[n]$ of size $\mathcal{O}(n/k \cdot \log m)$ that nontrivially intersects every set.

Node Selection (NS)
INPUT: A bipartite graph $G = (U, V, E)$ with $\deg(u) \in [d, \alpha \cdot d]$ for every $u \in U$.
OUTPUT: A set of $\mathcal{O}(|V|/d \cdot \log |U|)$ nodes from V such that every node in U has at least one selected neighbor but $\mathcal{O}(\alpha \cdot \log |U|)$ such selected neighbors.

To reduce finding anchors to an instance of the NS problem, we first build a bipartite graph G in which the nodes on the left correspond to strings $S \in \mathcal{S}$, the nodes on the right correspond to distinct length-ℓ substrings of P, and there is an edge connecting a node corresponding to a length-ℓ string H with a node corresponding to a string S when H occurs in S. Using suffix links, we can find the node of the suffix tree corresponding to every length-ℓ substring of S in $\mathcal{O}(|S|)$ total time, so the whole construction takes $\mathcal{O}(m \log m + \sum_{S \in \mathcal{S}} |S|) = \mathcal{O}(m \log m + N)$ time. The size of G is $\mathcal{O}(m + N)$, and the degree of every node on its left belongs to $[1/8 \cdot \ell, 1/4 \cdot \ell)$. We further partition G into a logarithmic number of graphs $G_0, G_1, \ldots, G_{\log m}$ where G_k contains all nodes v on the right of G such that the number of occurrences in P of the corresponding length-ℓ string belongs to $[2^k, 2^{k+1})$. For every node u on the left of G we find k such that at least $1/8 \cdot \ell / \log m$ of its neighbors exist in G_k, add u as a node on the left of G_k, and declare $\mathcal{S}_k$ to consist of all strings $S \in \mathcal{S}$ corresponding to nodes on the left of G_k. By construction, every $S \in \mathcal{S}$ corresponds to a node on the left of exactly one G_k, so we indeed obtain a partition of $\mathcal{S}$. For every $\mathcal{S}_k$ we solve the corresponding instance of the NS problem to obtain its corresponding set of anchors. We can assume that all strings in $\mathcal{S}_k$ are distinct, so there are at most m^2 nodes on the left of G_k, the degree of each such node belongs to $[1/8 \cdot \ell / \log m, 1/4 \cdot \ell]$ and, denoting by m_k the total number of occurrences in P of strings corresponding to nodes on the right of G_k, we have $\sum_k m_k \leq m$ and there are at most $m_k/2^k$ nodes on the right of G_k. At most

$\mathcal{O}((m_k/2^k)/(\ell/\log m)\cdot\log m)$ nodes on the right of G_k are designated as anchors, making the total number of occurrences of all anchors $\mathcal{O}(m/\ell\cdot\log^2 m)$. Also, every $S\in\mathcal{S}_k$ contains an occurrence of at least one anchor, and no more than $\mathcal{O}(\log^2 m)$ such occurrences.

It is not immediately clear that an instance of the NS problem always has a solution. We show that indeed it does, and that it can be efficiently found with a Las Vegas algorithm.

▶ **Lemma 10.** *A solution to an instance of the NS problem always exists and can be found in expected linear time.*

Proof. We independently choose each node of V with probability p to obtain the set X of selected nodes. Then, we check if the size of X is small enough, every node in U has at least one selected neighbor, and $\mathcal{O}(\alpha\cdot\log|U|)$ such selected neighbors. All these checks can be made in linear time in the size of the graph, so to show that a solution exists and can be found in expected linear time, it remains to show that we can adjust p to make the probability of failure equal to a constant less than 1. The expected size of X is obviously $p|V|$, so by Markov's inequality the probability that $|X|>4p|V|$ is at most $1/4$. The probability that a node in U has no neighbors in X is at most $(1-p)^d$. Thus, by union bound the probability that there exists at least one such node is at most $|U|\cdot(1-p)^d\le|U|\cdot e^{-pd}$. Consider a node in U of degree $d'\in[d,\alpha\cdot d]$. Its expected number of selected neighbors is pd'. Thus, by Chernoff's inequality the probability that its number of selected neighbors exceeds $(1+\delta)pd'$ is at most $e^{-\frac{\delta^2}{2+\delta}pd'}$. By setting $\delta=1$, we obtain that the probability of the number of selected neighbors exceeding $2p\alpha d$ is at most $e^{-\frac{1}{3}\alpha pd}$. By union bound, the probability that this happens for at least one node is at most $|U|\cdot e^{-\frac{1}{3}\alpha pd}$.

We choose $p=3\ln(4|U|)/d$. Then, the probability that the size of X exceeds $4p|V|=12\ln(4|U|)/d\cdot|V|=\mathcal{O}(\frac{|V|}{d}\cdot\log|U|)$ is at most $1/4$, the probability that there exists a node in U with no selected neighbor is at most $|U|\cdot e^{-pd}\le 1/4$, and the probability that there exists a node in U with more than $2p\alpha d=\mathcal{O}(\alpha\cdot\log|U|)$ selected neighbors is at most $|U|\cdot e^{-\frac{1}{3}\alpha pd}\le 1/4$, thus the overall probability of failure is at most $3/4$ as required. ◀

In the rest of this section we explain how to compute the bit vector V from the bit vector U after having obtained a set $\mathcal{A}$ of anchors for a set of strings $\mathcal{S}_k$ of total length N_k. For any $S\in\mathcal{S}_k$, since S contains an occurrence of at least one anchor $H\in\mathcal{A}$, for concreteness $S[j\,..\,(j+|H|-1)]=H$, any occurrence of S in P can be generated by choosing some occurrence of H in P, say $P[i\,..\,(i+|H|-1)]=H$, and then checking that $S[1\,..\,(j-1)]=P[(i-j+1)\,..\,(i-1)]$ and $S[(j+|H|)\,..\,|S|]=P[(i+|H|)\,..\,(i+|S|-j)]$. In other words, $S[1\,..\,(j-1)]$ should be a suffix of $P[1\,..\,(i-1)]$ and $S[(j+|H|)\,..\,|S|]$ should be a prefix of $P[(i+|H|)\,..\,|P|]$. In such case, we say that the occurrence of S in P is generated by H. By the properties of $\mathcal{A}$, any occurrence of $S\in\mathcal{S}_k$ is generated by at least one but no more than $\mathcal{O}(\log^2 m)$ anchors. For every $H\in\mathcal{A}$ we create a separate data structure $D(H)$ responsible for setting $V[i+|S|-1]=1$ if $U[i-1]=1$ and $P[i\,..\,(i+|S|-1)]=S$ is generated by H. We separately describe what information is used to initialize each $D(H)$ and how it is later processed to update V.

Initialization. $D(H)$ consists of two compact tries $T(H)$ and $T^r(H)$. For every occurrence of H in P, denoted by $P[i\,..\,(i+|H|-1)]=H$, $T(H)$ should contain a leaf corresponding to $P[(i+|H|)\,..\,|P|]\$$ and $T^r(H)$ should contain a leaf corresponding to $(P[1\,..\,(i-1)])^r\$$, both decorated with position i. Additionally, $D(H)$ stores a list $L(H)$ of pairs of nodes (u,v), where $u\in T^r(H)$ and $v\in T(H)$. Each such pair corresponds to an occurrence of H in a string $S\in\mathcal{S}_k$, $S[j\,..\,(j+|H|-1)]=H$, where u is the node of $T^r(H)$ corresponding to $(S[1\,..\,(j-1)])^r\$$ and v is the node of $T(H)$ corresponding to $S[(j+|H|+1)\,..\,|S|]\$$. We claim that $D(H)$, for all H, can be constructed in $\mathcal{O}(m\log m+N_k)$ total time.

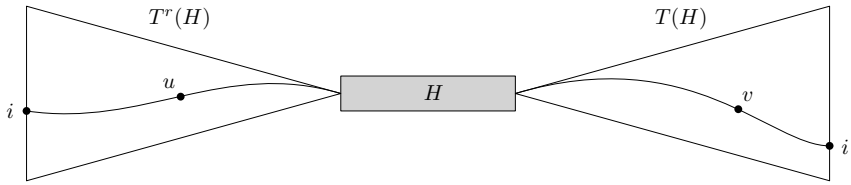

Figure 1 An occurrence of S starting at position i in P is generated by H: (u, v) corresponds to $S[j\,..\,(j+|H|-1)] = H$ and i appears in the subtree rooted at u as well as the subtree rooted at v.

We first construct the suffix tree ST of P and the suffix tree ST^r of P^r in $\mathcal{O}(m \log m)$ time. We augment both trees with a structure for answering *weighted ancestor* (WA) and *lowest common ancestor* (LCA) queries that are defined as follows. For a rooted tree T on n nodes with an integer weight $\mathcal{D}(v)$ assigned to every node u, such that the weight of the root is zero and $\mathcal{D}(u) < \mathcal{D}(v)$ if u is the parent of v, we say that a node v is a weighted ancestor of a node v at depth ℓ, denoted by $\mathrm{WA}_T(u, \ell)$, if v is the highest ancestor of u with weight of at least ℓ. Such queries can be answered in $\mathcal{O}(\log n)$ time after an $\mathcal{O}(n)$-time preprocessing [25]. For a rooted tree T, $\mathrm{LCA}_T(u, v)$ is the lowest node that is an ancestor of both u and v. Such queries can be answered in $\mathcal{O}(1)$ time after an $\mathcal{O}(n)$-time preprocessing [10]. Recall that every anchor H is represented by one of its occurrences in P. Using WA queries, we can access in $\mathcal{O}(\log m)$ time the nodes corresponding to H and H^r, respectively, and extract a lexicographically sorted list of suffixes following an occurrence of H in $P\$$ and a lexicographically sorted list of reversed prefixes preceding an occurrence of H in $P^r\$$ in time proportional to the number of such occurrences. Then, by iterating over the lexicographically sorted list of suffixes and using LCA queries on ST we can build $T(H)$ in time proportional to the length of the list, and similarly for $T^r(H)$. To construct $L(H)$ we start by computing, for every $S \in \mathcal{S}_k$ and $j = 1, \ldots, |S|$, the node of ST^r corresponding to $(S[1\,..\,j])^r$ and the node of ST corresponding to $S([(j+1)\,..\,|S|]$ (the nodes might possibly be implicit). Using suffix links this takes only $\mathcal{O}(|S|)$ time. We also find, for every length-ℓ substring $S[j\,..\,(j+\ell-1)]$ of S, an anchor $H \in \mathcal{A}$ such that $S[j\,..\,(j+\ell-1)] = H$, if any exists. This can be done by finding the nodes (implicit or explicit) of ST that correspond to the anchors, and then scanning over all length-ℓ substrings while maintaining the node of ST corresponding to the current substring using suffix links in $\mathcal{O}(|S|)$ total time. After having determined that $S[j\,..\,(j+\ell-1)] = H$ we add (u, v) to $L(H)$, where u and v are the previously found nodes of ST^r and ST corresponding to $(S[1\,..\,(j-1)])^r$ and $S[(j+\ell)\,..\,|S|]$, respectively. By construction, we have the following property, also illustrated in Figure 1.

▶ **Fact 11.** A string $S \in \mathcal{S}_k$ starts at position $i - j + 1$ in P if and only if, for some anchor $H \in \mathcal{A}$, $L(H)$ contains a pair (u, v) corresponding to $S[j\,..\,(j+|H|-1)] = H$, such that the subtree of $T^r(H)$ rooted at u and that of $T(H)$ rooted at v contain a leaf decorated with i.

Note that the overall size of all lists $L(H)$, when summed up over all $H \in \mathcal{A}$, is $\mathcal{O}(N_k/\ell \cdot \log^2 m)$, because any $S \in \mathcal{S}_k$ contains $\mathcal{O}(\log^2 m)$ occurrences of anchors, and since each S is of length at least ℓ, there are only $\mathcal{O}(N_k/\ell)$ strings in $\mathcal{S}_k$.

Processing. The goal of processing $D(H)$ is to efficiently process all occurrences generated by H. As a preliminary step, we decompose $T^r(H)$ and $T(H)$ into heavy paths. Then, for every pair of leaves $u \in T^r(H)$ and $v \in T(H)$ decorated by the same i, we consider all heavy paths above u and v. Let $p = u_1 - u_2 - \ldots$ be a heavy path above u in $T^r(H)$ and $q = v_1 - v_2 - \ldots$ be a heavy path above v in $T(H)$, where u_1 is the head of p and v_1 is the

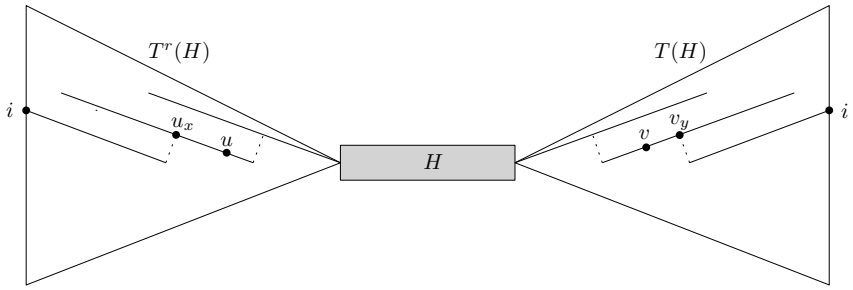

Figure 2 An occurrence of S starting at position i in P corresponds to a triple $(i, \mathcal{L}(u_x), \mathcal{L}(v_y))$ on some auxiliary list.

head of q, respectively. Further, choose the largest x such that u is in the subtree rooted at u_x, and the largest y such that v is in the subtree rooted at v_y (by the choice of p and q, u is in the subtree rooted at u_1 and v is in the subtree rooted at v_1, so this is well-defined). We add $(i, |\mathcal{L}(u_x)|, |\mathcal{L}(v_y)|)$ to an auxiliary list associated with the pair of heavy paths (p, q). In the rest of the processing we work with each such list separately. Notice that the overall size of all auxiliary lists, when summed up over all $H \in \mathcal{A}$, is $\mathcal{O}(m/\ell \cdot \log^4 m)$, because there are at most $\log^2 m$ pairs of heavy paths above u and v decorated by the same i, and the total number of leaves in all trees $T^r(H)$ and $T(H)$ is bounded by the total number of occurrences of all anchors in P, which is $\mathcal{O}(m/\ell \cdot \log^2 m)$. By Fact 11, there is an occurrence of a string $S \in \mathcal{S}_k$ generated by H and starting at position $i - j + 1$ in P if and only if $L(H)$ contains a pair (u, v) corresponding to $S[j\,..\,(j + |H| - 1)] = H$ such that, denoting by p the heavy path containing u in $T^r(H)$ and by q the heavy path containing v in $T(H)$, the auxiliary list associated with (p, q) contains a triple (i, x, y) such that $x \geq |\mathcal{L}(u)|$ and $y \geq |\mathcal{L}(v)|$. This is illustrated in Figure 2. From now on we focus on processing a single auxiliary list associated with (p, q) together with a list of pairs (u, v) such that u belongs to p and v belongs to q.

An auxiliary list can be interpreted geometrically: for every (i, x, y) we create a red point (x, y), and for every (u, v) we create a blue point $(|\mathcal{L}(u)|, |\mathcal{L}(v)|)$. Then, each occurrence of $S \in \mathcal{S}_k$ generated by H corresponds to a pair of points (p_1, p_2) such that p_1 is red, p_2 is blue, and p_1 dominates p_2. We further reduce this to a collection of simpler instances in which all red points already dominate all blue points. This can be done with a divide-and-conquer procedure which is essentially equivalent to constructing a 2D range tree [11]. The total number of points in all obtained instances increases by a factor of $\mathcal{O}(\log^2 m)$, making the total number of red points in all instances $\mathcal{O}(m/\ell \cdot \log^6 m)$, while the total number of blue points is $\mathcal{O}(N_k/\ell \cdot \log^4 m)$. There is an occurrence of a string $S \in \mathcal{S}_k$ generated by H and starting at position $i - j + 1$ in P if and only if some simpler instance contains a red point created for some (i, x, y) and a blue point created for some (u, v) corresponding to $S[j\,..\,(j + |H| - 1)] = H$. In the following we focus on processing a single simpler instance.

To process a simpler instance we need to check if $U[i - j] = 1$, for a red point created for some (i, x, y) and a blue point created for some (u, v) corresponding to $S[j\,..\,(j+|H|-1)] = H$, and if so set $V[i - j + |S|] = 1$. This has a natural interpretation as an instance of BMM: we create an $(5/4 \cdot \ell) \times (5/4 \cdot \ell)$ matrix M such that $M[|S| - j, 5/4 \cdot \ell + 1 - j] = 1$ if and only if there is a blue point created for some (u, v) corresponding to $S[j\,..\,(j+|H|-1)] = H$; then for every red point created for some (i, x, y) we construct a bit vector $U_i = U[(i - 5/4 \cdot \ell)\,..\,(i - 1)]$ (if $i < 5/4 \cdot \ell$, we pad U_i with 0s to make its length always equal to $5/4 \cdot \ell$); calculate $V_i = M \times U_i$; and finally set $V[i + j] = 1$ whenever $V_i[j] = 1$ (and $i + j \leq m$).

▶ **Lemma 12.** *$V_i[k] = 1$ if and only if there is a blue point created for some (u, v) corresponding to $S[j\,..\,(j + |H| - 1)] = H$ such that $U[i - j] = 1$ and $k = |S| - j$.*

The total length of all vectors U_i and V_i is $\mathcal{O}(m \log^6 m)$, so we can afford to extract the appropriate fragment of U and then update the appropriate fragment of V. The bottleneck is computing the matrix-vector product $V_i = M \times U_i$. Naïvely, this might take $\mathcal{O}(N_k/\ell \cdot \log^4 m)$ time, because the total number of 1s in all matrices M is bounded by the total number of blue points. We overcome this by processing together all multiplications concerning the same matrix M. Let $U_{i_1}, U_{i_2}, \ldots, U_{i_s}$ be all bit vectors that need to be multiplied with M, and z a parameter to be determined later. We distinguish between two cases: (i) If $\boldsymbol{s < z}$ we compute the products naïvely by iterating over all 1s in M, and the total computation time, when summed up over all such matrices M, is $\mathcal{O}(N_k/\ell \cdot \log^4 m \cdot z)$; (ii) If $\boldsymbol{s \geq z}$ we partition the bit vectors into $\lceil s/z \rceil \leq s/z + 1$ groups of z (padding the last group with bit vectors containing all 0s). For every group, we create a single matrix whose columns contain all the bit vectors belonging to the group. Thus, we reduce computing all matrix-vector products $M \times U_i$ to computing $\mathcal{O}(s/z)$ matrix-matrix products of the form $M \times M'$, where M' is an $(5/4 \cdot \ell) \times z$ matrix. M' is not necessarily a square matrix, but we can still apply the fast matrix multiplication algorithm to compute $M \times M'$ using the standard trick of decomposing the matrices into square blocks.

▶ **Lemma 13.** *If two $\mathcal{N} \times \mathcal{N}$ matrices can be multiplied in $\mathcal{O}(\mathcal{N}^\omega)$ time, then, for any $\mathcal{N} \geq \mathcal{N}'$, an $\mathcal{N} \times \mathcal{N}$ and an $\mathcal{N} \times \mathcal{N}'$ matrix can be multiplied in $\mathcal{O}((\mathcal{N}/\mathcal{N}')^2 \mathcal{N}'^\omega)$ time.*

By applying Lemma 13, we can compute $M \times M'$ in $\mathcal{O}(\ell^2 z^{\omega-2})$ time (as long as we later verify that $5/4 \cdot \ell \geq z$), so all products $M \times U_i$ can be computed in $\mathcal{O}(\ell^2 z^{\omega-2} \cdot (s/z + 1))$ time. Note that this case can occur only $\mathcal{O}(m/(\ell \cdot z) \cdot \log^6 m)$ times, because all values of s sum up to $\mathcal{O}(m/\ell \cdot \log^6 m)$. This makes the total computation time, when summed up over all such matrices M, $\mathcal{O}(\ell^2 z^{\omega-2} \cdot m/(\ell \cdot z) \cdot \log^6 m) = \mathcal{O}(\ell z^{\omega-3} \cdot m \log^6 m)$.

▶ **Theorem 14.** *An instance of the AP problem where all strings are of type 1 can be solved in expected $\mathcal{O}(m^{1.373} + N)$ time.*

Proof. The total time complexity is first $\mathcal{O}(m + N)$ to construct the graph G and then all graphs G_k, then expected linear time to solve their corresponding instances of the NodeSelection problem, partition $\mathcal{S}$ into $\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_{\log m}$ and obtain their corresponding sets of anchors H. The time to initialize all structures $D(H)$ is $\mathcal{O}(m \log m + N_k)$. For every $D(H)$, we obtain in $\mathcal{O}(m/\ell \cdot \log^6 m + N_k/\ell \cdot \log^4 m)$ time a number of simpler instances, and then construct the corresponding Boolean matrices M and bit vectors U_i in additional $\mathcal{O}(m \log^6 m)$ time. Note that some M might be sparse, so we need to represent them as a list of 1s. Then, summing up over all matrices M and both cases, we spend $\mathcal{O}(N_k/\ell \cdot \log^4 m \cdot z + \ell z^{\omega-3} \cdot m \log^6 m)$ time. We would like to assume that $\ell \geq \log^4 m$ so that we can set $z = \ell/\log^4 m$. This is indeed possible, because for any t we can switch to a more naïve approach to process all strings of length at most t in $\mathcal{O}(mt^2 + N_k)$ time: for every possible length $t' \leq t$ scan P with a window of length t', in every step check if the current window $P[i\,..\,(i+t'-1)]$ corresponds to a string $S \in \mathcal{S}$, if so and $U[i-1] = 1$ then set $V[i+t'-1] = 1$. After applying this with $t = \log^4 m$ in $\mathcal{O}(m \log^8 m + N_k)$ time, we can set $z = \ell/\log^4 m$ (so that indeed $5/4 \cdot \ell \geq z$ as required in case $s \geq z$) and the overall time complexity for all matrices M and both cases becomes $\mathcal{O}(N_k + \ell^{\omega-2} \cdot m \log^{6+4(3-\omega)} m)$. Summing up over all values of k and ℓ and taking the initialization into account we obtain $\mathcal{O}(m \log^8 m + m^{\omega-1} \log^{7+4(3-\omega)} m + N) = \mathcal{O}(m^{1.373} + N)$ total time. (We hide $\log^{\mathcal{O}(1)} m$ factors using the fact that $\omega < 2.373$ [30, 54]). ◀

4.2 Wrapping Up

In the full version of the paper we design $\mathcal{O}(m^{1.373} + N)$ and $\mathcal{O}(m^{1.381} + N)$-time algorithms for an instance of the AP problem where all strings are of type 2 and 3, respectively. For type 2 strings, we follow the same ideas as for type 1 strings, except that instead of the NodeSelection problem we select the anchors by applying a periodicity-based argument. For type 3 strings, we need both fast matrix multiplication and FFT. Together with Theorem 14, this gives us Theorem 2.

References

1 A. Abboud, A. Backurs, and V.V. Williams. If the Current Clique Algorithms are Optimal, So is Valiant's Parser. In *56th IEEE Symposium on Foundations Of Computer Science (FOCS)*, pages 98–117, 2015.

2 A. Abboud and V.V. Williams. Popular Conjectures Imply Strong Lower Bounds for Dynamic Problems. In *55th IEEE Symposium on Foundations Of Computer Science (FOCS)*, pages 434–443, 2014.

3 K. Abrahamson. Generalized String Matching. *SIAM J. Comput.*, 16(6):1039–1051, 1987.

4 M. Alzamel, L.A.K. Ayad, G. Bernardini, R. Grossi, C.S. Iliopoulos, N. Pisanti, S.P. Pissis, and G. Rosone. Degenerate string comparison and applications. In *18th International Workshop on Algorithms in Bioinformatics (WABI)*, volume 113 of *LIPIcs*, pages 21:1–21:14, 2018.

5 A. Amir, M. Lewenstein, and E. Porat. Faster algorithms for string matching with k mismatches. *J. Algorithms*, 50(2):257–275, 2004.

6 K. Aoyama, Y. Nakashima, T. I, S. Inenaga, H. Bannai, and M. Takeda. Faster Online Elastic Degenerate String Matching. In *29th Symposium on Combinatorial Pattern Matching (CPM)*, volume 105 of *LIPIcs*, pages 9:1–9:10, 2018.

7 V.L. Arlazarov, E.A. Dinic, M.A. Kronrod, and I.A. Faradžev. On economical construction of the transitive closure of a directed graph. *Soviet Mathematics Doklady*, 11(5):1209–1210, 1970.

8 A. Backurs and P. Indyk. Which Regular Expression Patterns Are Hard to Match? In *57th IEEE Symposium on Foundations Of Computer Science (FOCS)*, pages 457–466, 2016.

9 N. Bansal and R. Williams. Regularity Lemmas and Combinatorial Algorithms. In *50th IEEE Symposium on Foundations Of Computer Science (FOCS)*, pages 745–754, 2009.

10 M.A. Bender and M. Farach-Colton. The LCA Problem Revisited. In *4th Latin American symposium on Theoretical INformatics (LATIN)*, volume 1776 of *Springer LNCS*, pages 88–94, 2000.

11 J.L. Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM*, 18(9):509–517, 1975.

12 G. Bernardini, N. Pisanti, S.P. Pissis, and G. Rosone. Pattern Matching on Elastic-Degenerate Text with Errors. In *24th International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 74–90, 2017.

13 K. Bringmann, F. Grandoni, B. Saha, and V.V. Williams. Truly Sub-cubic Algorithms for Language Edit Distance and RNA-Folding via Fast Bounded-Difference Min-Plus Product. In *56th IEEE Symposium on Foundations Of Computer Science (FOCS)*, pages 375–384, 2016.

14 K. Bringmann, A. Grønlund, and K.G. Larsen. A Dichotomy for Regular Expression Membership Testing. In *58th IEEE Symposium on Foundations Of Computer Science (FOCS)*, pages 307–318, 2017.

15 T.M. Chan. Speeding Up the Four Russians Algorithm by About One More Logarithmic Factor. In *26th ACM-SIAM Symposium On Discrete Algorithms (SODA)*, pages 212–217, 2015.

16 Y.-J. Chang. Hardness of RNA Folding Problem With Four Symbols. In *27th Symposium on Combinatorial Pattern Matching (CPM)*, volume 54 of *LIPIcs*, pages 13:1–13:12, 2016.

17 K. Chatterjee, B. Choudhary, and A. Pavlogiannis. Optimal Dyck Reachability for Data-dependence and Alias Analysis. In *45th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*, pages 30:1–30:30, 2018.

18 A. Cisłak, S. Grabowski, and J. Holub. SOPanG: online text searching over a pan-genome. *Bioinformatics*, page bty506, 2018.

19 P. Clifford and R. Clifford. Simple deterministic wildcard matching. *Inf. Process. Lett.*, 101(2):53–54, 2007.

20 R. Cole and R. Hariharan. Verifying candidate matches in sparse and wildcard matching. In *34th ACM Symposium on Theory Of Computing (STOC)*, pages 592–601, 2002.

21 The Computational Pan-Genomics Consortium. Computational pan-genomics: status, promises and challenges. *Briefings in Bioinformatics*, 19(1):118–135, 2018.

22 M. Crochemore, C. Hancart, and T. Lecroq. *Algorithms on strings*. Cambridge University Press, 2007.

23 M. Crochemore and D. Perrin. Two-Way String Matching. *J. ACM*, 38(3):651–675, 1991.

24 A. Czumaj and A. Lingas. Finding a Heaviest Vertex-Weighted Triangle Is not Harder than Matrix Multiplication. *SIAM J. Comput.*, 39(2):431–444, 2009.

25 M. Farach and S. Muthukrishnan. Perfect Hashing for Strings: formalization and Algorithms. In *7th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 1075 of *Springer LNCS*, pages 130–140, 1996.

26 N.J. Fine and H.S. Wilf. Uniqueness Theorems for Periodic Functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965.

27 M.J. Fischer and A.R. Meyer. Boolean Matrix Multiplication and Transitive Closure. In *12th IEEE Symposium on Switching and Automata Theory (SWAT/FOCS)*, pages 129–131, 1971.

28 M.J. Fischer and M.S. Paterson. String matching and other products. In *7th SIAM-AMS Complexity of Computation*, pages 113–125, 1974.

29 M.E. Furman. Application of a method of fast multiplication of matrices in the problem of finding the transitive closure of a graph. *Soviet Mathematics Doklady*, 11(5):1252, 1970.

30 F. Le Gall. Powers of tensors and fast matrix multiplication. In *39th International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 296–303, 2014.

31 P. Gawrychowski and P. Uznański. Towards Unified Approximate Pattern Matching for Hamming and L_1 Distance. In *45th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 107 of *LIPIcs*, pages 62:1–62:13, 2018.

32 R. Grossi, C.S. Iliopoulos, C. Liu, N. Pisanti, S.P. Pissis, A. Retha, G. Rosone, F. Vayani, and L. Versari. On-Line Pattern Matching on Similar Texts. In *28th Symposium on Combinatorial Pattern Matching (CPM)*, volume 78 of *LIPIcs*, pages 9:1–9:14, 2017.

33 M. Henzinger, S. Krinninger, D. Nanongkai, and T. Saranurak. Unifying and Strengthening Hardness for Dynamic Problems via the Online Matrix-Vector Multiplication Conjecture. In *47th ACM Symposium on Theory Of Computing (STOC)*, pages 21–30, 2015.

34 J. Holub, W.F. Smyth, and S. Wang. Fast pattern-matching on indeterminate strings. *J. Discrete Algorithms*, 6(1):37–50, 2008.

35 C.S. Iliopoulos, R. Kundu, and S.P. Pissis. Efficient Pattern Matching in Elastic-Degenerate Texts. In *11th International Conference on Language and Automata Theory and Applications (LATA)*, volume 10168 of *Springer LNCS*, pages 131–142, 2017.

36 P. Indyk. Faster Algorithms for String Matching Problems: Matching the Convolution Bound. In *39th Symposium on Foundations Of Computer Science (FOCS)*, pages 166–173, 1998.

37 A. Itai and M. Rodeh. Finding a Minimum Circuit in a Graph. In *9th ACM Symposium on Theory Of Computing (STOC)*, pages 1–10, 1977.

38 IUPAC-IUB Commission on Biochemical Nomenclature. Abbreviations and symbols for nucleic acids, polynucleotides, and their constituents. *Biochemistry*, 9(20):4022–4027, 1970.

39 A. Kalai. Efficient pattern-matching with don't cares. In *13th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 655–656, 2002.

40 D.E. Knuth, J.H. Morris Jr., and V.R. Pratt. Fast Pattern Matching in Strings. *SIAM J. Comput.*, 6(2):323–350, 1977.

41 T. Kociumaka, J. Radoszewski, W. Rytter, and T. Waleń. Internal Pattern Matching Queries in a Text and Applications. In *26th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 532–551, 2015.

42 T. Kopelowitz and R. Krauthgamer. Color-distance oracles and snippets. In *27th Symposium on Combinatorial Pattern Matching (CPM)*, volume 54 of *LIPIcs*, pages 24:1–24:10, 2016.

43 K.G. Larsen, I. Munro, J.S. Nielsen, and S.V. Thankachan. On hardness of several string indexing problems. *Theor. Comput. Sci.*, 582:74–82, 2015.

44 L. Lee. Fast Context-free Grammar Parsing Requires Fast Boolean Matrix Multiplication. *J. ACM*, 49(1):1–15, 2002.

45 J. Matoušek. Computing Dominances in E^n. *Inf. Process. Lett.*, 38(5):277–278, 1991.

46 I. Munro. Efficient Determination of the Transitive Closure of a Directed Graph. *Inf. Process. Lett.*, 1(2):56–58, 1971.

47 G. Navarro. NR-grep: a fast and flexible pattern-matching tool. *Softw., Pract. Exper.*, 31(13):1265–1312, 2001.

48 S.P. Pissis and A. Retha. Dictionary Matching in Elastic-Degenerate Texts with Applications in Searching VCF Files On-line. In *17th International Symposium on Experimental Algorithms (SEA)*, volume 103 of *LIPIcs*, pages 16:1–16:14, 2018.

49 L. Roditty and U. Zwick. On Dynamic Shortest Paths Problems. In *12th European Symposium on Algorithms (ESA)*, volume 3221 of *Springer LNCS*, pages 580–591, 2004.

50 M. Ružić. Constructing Efficient Dictionaries in Close to Sorting Time. In *35th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 5125 of *Springer LNCS*, pages 84–95, 2008.

51 D.D. Sleator and R.E. Tarjan. A Data Structure for Dynamic Trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.

52 L.G. Valiant. General Context-free Recognition in Less Than Cubic Time. *J. Comput. Syst. Sci.*, 10(2):308–315, April 1975.

53 P. Weiner. Linear Pattern Matching Algorithms. In *14th IEEE Annual Symposium on Switching and Automata Theory (SWAT/FOCS)*, pages 1–11, 1973.

54 V.V. Williams. Multiplying matrices faster than Coppersmith-Winograd. In *44th ACM Symposium on Theory Of Computing Conference (STOC)*, pages 887–898, 2012.

55 V.V. Williams and R. Williams. Finding a maximum weight triangle in $n^{3-\delta}$ time, with applications. In *38th ACM Symposium on Theory Of Computing Conference (STOC)*, pages 225–231, 2006.

56 V.V. Williams and R. Williams. Subcubic equivalences between path, matrix and triangle problems. In *51st IEEE Symposium on Foundations Of Computer Science (FOCS)*, pages 645–654, 2010.

57 S. Wu and U. Manber. Agrep – A Fast Approximate Pattern-Matching Tool. In *USENIX Technical Conference*, pages 153–162, 1992.

58 H. Yu. An improved combinatorial algorithm for Boolean matrix multiplication. *Inf. Comput.*, 261(Part):240–247, 2018.

59 U. Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002.